

**APPLICATION FOR  
UNITED STATES PATENT**

in the name of

**Karl Haakonsen**

of

**Medical Learning Co., Inc.**

for

**MORPHING**

**Kenneth F. Kozik**  
Fish & Richardson P.C.  
225 Franklin Street  
Boston, MA 02110-2804  
Tel.: (617) 542-5070  
Fax: (617) 542-8906

**ATTORNEY DOCKET:**

**11328-006001**

**DATE OF DEPOSIT:**

June 26, 2000

**EXPRESS MAIL NO.:**

**EL** 485521601 **US**

~~MORPHING~~

Priority Claim

This application claims priority from co-pending provisional application Serial No. ~~60/165291~~,<sup>1A1</sup> filed on ~~November 12, 1999~~, which is incorporated by reference herein.

Reference to Related Applications

This application relates to the following applications  
U.S. Ser. No. \_\_\_\_\_, entitled "Virtual Patient Hot Spots",  
filed \_\_\_\_\_; U.S. Ser. No. \_\_\_\_\_, entitled "Web Simulator",  
10 filed \_\_\_\_\_; U.S. Ser. No. \_\_\_\_\_, entitled "Patient  
Simulator", filed \_\_\_\_\_.

Background

Desktop computer programs such as flight simulators,  
15 word-processors, and spreadsheets quickly respond to user  
input. Providing this kind of inter-activity over the  
Internet, however, has posed something of a challenge. This  
challenge stems, in part, from the simple communication model  
that describes most Internet traffic: clients (e.g., web-  
20 browsers) request pre-written web pages from servers, and the  
servers send the requested pages back for display.

Programmers have developed a number of different  
techniques to make web pages more inter-active. For example,  
some web pages include, or refer to, programs known as  
25 "applets." When a browser receives a web-page featuring an  
applet, the browser executes the applet instructions, for  
example, to receive user input, change a browser GUI  
(Graphical User Interface) display, and communicate with the  
server providing the web-page. Some applets, however, include  
30 a large number of instructions and can require a significant

amount of time to travel over the Internet to a user's browser.

Another technique for making web pages more responsive to users involves dynamic generation of web pages by servers.

5 For example, PHP (Personal Home Page), ASP (Active Server Page), and CGI (Common Gateway Interface) scripts can dynamically produce a web page based on script instructions and variables. Script processing, however, adds yet another task to web-servers faced with handling large bursts of  
10 browser requests for information.

#### Summary

In an aspect, the invention features a method of displaying an image of at least a portion of a virtual patient  
15 including accessing identification of a video file, the video file comprising video data that depicts virtual patient features over a range of the features, determining an offset , into the video file and presenting the video image corresponding to the offset.

20 Advantages of the invention will become apparent in view of the following description, including the figures, and the claims.

#### Brief Description of the Drawings

25 FIGs. 1 to 5 are diagrams illustrating a network application architecture.

FIG. 6 is a flowchart of a server process for providing an application to different network users.

FIG. 7 is a flowchart of a client process for presenting  
30 a user interface to network users.

FIGs. 8-14 are screenshots of a user interface that presents a virtual patient.

FIGs. 15A and 15B are listings of statements associating multimedia files with different virtual patient characteristics.

5 FIG. 16 is a flowchart of a process for providing a virtual patient.

FIG. 17 is a diagram of a virtual patient simulator.

FIG. 18 is a flowchart of a process for evolving a medical condition.

FIG. 19 is a listing of statements defining an evolution.

10 FIG. 20 is a listing of statements defining a migration.

FIG. 21 is a flowchart of a process for changing multimedia presentations associated with a virtual patient.

FIG. 22 is a diagram illustrating a change of a multimedia presentation associated with a virtual patient.

15 FIG. 23 is a flowchart of a process for morphing a virtual patient.

FIG. 24 is a listing of statements defining a morphing operation.

20 FIG. 25 is a flowchart of a process for providing virtual patient responses to questions.

FIG. 26 is a listing of statements defining virtual patient answers to questions.

FIG. 27 is a listing of statements defining a response to a lab test.

25 FIG. 28 is a listing of statements defining an action.

FIG. 29 is a listing of statements defining a response to physical examination.

FIG. 30 is a listing of statements defining a computation.

30 FIG. 31 is a diagram of a virtual patient simulator using the network application architecture.

Like reference symbols in the various drawings indicate like elements.

## Detailed Description

5

### I. Network Architecture

#### A. Introduction

FIG. 1 shows a system 100 that uses the Internet 104 as a  
10 vehicle for providing complex, interactive applications to a large number of network users operating ordinary web-browsers 102a, 102b (e.g., Microsoft™ Internet Explorer™). The architecture 100 provides each user with a different "instance" of the application. That is, each user perceives  
15 an application program responsive to that user's input, much like an ordinary program residing on the user's personal computer. In the scheme shown in FIG. 1, the web-browsers 102a-102b need not receive the actual application instructions, but instead receive interface instructions for  
20 constructing a browser display and presenting different user interface controls. The interfaces are like application facades with the real application instructions residing on a remote system.

As shown in FIG. 1, the system 100 includes a network  
25 server 106 that stores the "state" 112 of each application instance. State data 112 can include the current values of different variables used by an application. A single application can handle more than one "instance." For example, an application can read an instance's state data 112, perform  
30 some processing, update and store the instance's state data 112, and move on to another instance's state data.

Conceptually, state data 112 represents the intersection between local area network computers 118a-118c running different copies of an application 120a-120d and a process 108 that dynamically constructs web pages for each application instance. Applications 120a-120d continually operate and update the state data 112. Independently, the server 106 can process user input and prepare browser instructions (e.g., HTML (Hypertext Markup Language)) for each instance.

For example, in one embodiment, the applications 120a-120d provide users with a simulation of a patient medical exam (i.e., simulation of a medical exam to a "virtual" patient). The simulation enables network users to interact with web-page controls (e.g., buttons, icons, and text fields) to examine, diagnose, and attempt to return a virtual patient to health. For this application, the state data 112 can include a virtual patient's vital signs (e.g., heart rate, pulse, and weight), images or references to images of the virtual patient's current appearance, medical orders and queries received from the user, and other information. The medical simulation applications 120a-120d continually update the state data 112 for a patient to reflect the virtual patient's health. For example, the medical simulation applications 120a-120d may slowly decrease a virtual patient's weight over time when a user prescribes a diet. While the medical simulation applications 120a-120d run behind the scenes, an interface generator 110 uses the state data 112 to construct browser instructions that display an appearance of the user's virtual patient and provide information (e.g., a medical history) requested by the user.

The network system of FIG. 1 can handle the large numbers of users that typically visit popular web sites. For example, pooling the computational resources of the LAN computers 118a-

118c enables the system 100 to support a large number of application instances. Additionally, the system 100 enables a network administrator to add computers to a local area network to further increase system "horse-power". Additionally, the server 106 does not need to deliver copies of the application 120a-120d to each network user. This can reduce network traffic. Keeping the application instructions on local area network machines also enables a site manager to easily replace and modify the applications, for example, to fix bugs or provide new features. This also prevents users from easily copying the programs. The system 100 also separates interface generation (server 106) from the actual "number crunching" (i.e., execution) of the application 120a-120d. This separation enables the server 106 to easily customize each user's interface. For example, more sophisticated computers or computers featuring faster network connections can receive interface instructions for more elaborate multimedia presentations.

#### B. Illustration of Operation

FIGs. 2-4 illustrate operation of the system 100. In FIG. 2, a user navigates browser 102a to the server 106, for example, by specifying the server's URL (Universal Resource Locator) (e.g., [www.medicalsimulator.com](http://www.medicalsimulator.com)). In response, as shown in FIG. 3, the server 106 initializes state data 112 for a new application instance. Initializing the state data 112 may include generating an identifier for the instance, for example, an identifier based on the IP (Internet Protocol) address of the user. The server 106 also selects an application 120a-120d to handle the instance. For example, the server 106 may select a particular application 120a-120d based on the current load of the different applications (e.g.,

load-balance based on how many different instances each application currently handles), the speed of the local area network computer running the application, and so forth.

As shown in FIG. 4, after initialization, communication with the user and execution of the application can proceed independently of one another, for the most part. For example, an application 120b can read and update state data 112 regardless of how frequently the server 106 transmits or receives data from application 120a. Thus, even if a user directs their browser to a different site, the application instance can persist and, potentially, continue. To continue the patient simulation example, even though a user visits a different site to check on stock prices, their patient's ailment will progress. The system does not strictly impose this independence. For example, the server 106 can control the application speed based on the speed of a users connection or how frequently the user interacts with an interface. The server 106 can also store the state data 112 to freeze an application instance for later use.

The server process 108 that handles user interaction can communicate with the "back-end" via the server 106 database 112. For example, the process 108 and application 120b can communicate using a technique known as "handshaking." Thus, a user may interact with an interface control (e.g., a form field on an HTML page) displayed by their browser to send user input to the server 106. The server 106 can write this input into the state data 112 for the user's application instance and set a handshake flag indicating the input requires application 120b processing. After the application 120b finishes its processing it can reset the handshake flag. The interface generator 110 may wait for the application 120b to reset the handshake flag before constructing a new interface.



State data 112 may be distributed across different LAN computers 108a-108c instead of residing solely on the server 106. For example, as shown in FIG. 5, each application copy may maintain information for each instance locally. The interface generator 110 can retrieve the state data 112 from the LAN computers when constructing an interface.

### C. Client/Server Operation

FIG. 6 shows a flowchart of a server operation 130. As shown, the server identifies 132 different applications running on the LAN computers. For example, the server may run a process or thread that establishes a connection (e.g., a "socket" connection) with each application that announces itself to the server. Though shown in FIG. 6 as initial activity of the server, identifying 132 available applications on the LAN computers can continue over time as different applications and LAN computers go on and off-line.

After receiving 134 a request from a user for an instance of an application, the server 106 selects 136 an application to handle the instance and initializes 138 the state data for that instance. Thereafter, the application can read and/or update the state data, for example, by issuing database commands over an ODBC (Open Database Connectivity) connection. The server also uses the state data to generate interface instructions that control the appearance of the application on a user's browser 142, 144.

The server 106 can construct an interface using a variety of techniques. For example, the interface generator can use PHP (Personal Home Page), ASP (Active Server Page), or CGI (Common Gateway Interface) scripts to dynamically generate HTML or XML (Extensible Markup Language) interface instructions. Typically, these pages will include

instructions that read state data for an instance (e.g., by looking up the instance identifier), and, based on the state data values, construct a page of sounds, graphics, and user interface controls. The user interface controls may include "form field" controls and/or a "submit" button that receives user input and transmits 146 this input to the server for processing (e.g., [www.medicalsimulator.com?action=CPR](http://www.medicalsimulator.com?action=CPR)). The interface may also include instructions that periodically issue a request to the server 106 for an update.

As shown in FIG. 7, a browser (or other client) receives and processes the generated instructions or data 152 to present the specified pictures, text, or sounds to a user. Though the browser receives discrete sets of web-page instructions, the resulting sequence of displays is nearly indistinguishable from that of an ordinary desktop application. The instructions may also feature applets 154, ActiveX controls, or other programmatic instructions. These instructions can provide fast, direct updating of multimedia elements of an interface. For example, instead of requesting reconstruction of an entire new set of interface instructions, an applet may establish a JDBC (Java Database Connectivity) connection to the data stored on the server 106 and continually replace an image on the interface with a new image. For example, the applet may quickly present a series of images of a patient when the patient's health rapidly improves.

The system 100 described in FIGs. 1-7 can be used to provide a wide variety of applications. For example, the system 100 can be used to provide a simulation of a virtual patient to different users on the Internet.

## II. Virtual Patient Simulation

## A. User Interface

FIGs. 8-14 show screenshots of a user interface presented by a virtual patient simulator. The simulator provides an interactive, multimedia simulation of a medical examination.

5 The user, acting as a doctor, can examine and interview the virtual patient, track vital signs, enter orders and recommendations, and conduct other simulated activities. Chronic and acute conditions evolve and disappear over time as the virtual patient's health responds to a user's treatments  
10 or lack thereof.

The simulator provides extensive medical education and reference information such as pharmacological references (e.g., PDR (Physician's Desk Reference) notes), videos and images illustrating proper use of products and procedures,  
15 medical dictionary entries (e.g., from Gray's Anatomy), and links to medical information on the Internet.

The virtual patient and reference information provide an engaging education experience for medical students, medical professionals seeking continuing medical education, and other  
20 consumers. The simulation enables users to experiment with different patient treatments and quickly witness the impact of their recommendations on the virtual patient.

As shown in FIG. 8, a user interface 800 provides a graphic model 802 of a virtual patient. The simulation  
25 changes depiction of the model 802 based on the virtual patient's state. For example, the model image may show the hips of the model 802 growing as the virtual patient's weight increases. Similarly, the model's 802 complexion color may sallow due to some illness.

30 Interface 800 controls 808a-808b enable a user to view both the front and back sides of the virtual patient. Other controls 808c-808d enable the user to select whether the

virtual patient model 802 appears clothed or unclothed. The user interface 800 also displays vital statistics 806 such as blood pressure, pulse rate, temperature and so forth.

As shown, the interface 800 provides a palette 810 of controls 810a-810h that correspond to different examination techniques. As shown, the palette 810 includes controls 810a-810h that present a cardiology EKG 810a of the virtual patient, representations of the virtual patient's extremities 810c (e.g., hands and feet), radiology images 810d, microscopy images 810f, and neurological information 810e such as images showing the virtual patient's gait or a "recording" of the virtual patient's voice.

As shown, a user has selected a virtual stethoscope tool 810g from the palette 810. For virtual stethoscope 810g use, the interface 800 presents front 812 and back 814 images of the virtual patient's torso. As shown, the interface 800 also displays "hot spots" 816, 818, and 820. Each hot spot 816a-816d, 818a-818d, 820a-820d maps to a particular location on the virtual patient's torso. Selecting (e.g., using a mouse) one of the hot spots 816a-816d, 818a-818d, 820a-820d simulates placement of a stethoscope on the hot spot and produces corresponding sounds on a computer's audio output (e.g., speakers). Just as in a real-life medical exam, use of the virtual stethoscope can identify breathing or cardiac abnormalities. For example, using the virtual stethoscope to listen at hot spots 816a-816d produces sounds of the virtual patient's heart. More particularly, hot spot 816a corresponds to the second right intercostal space. Thus, selecting hot spot 816a causes the simulator to produce sounds corresponding to the virtual patient's aortic valve. Other hot spots "listen" to other bodily functions. For example, hot spots 818a-818d correspond to the virtual patient's renal arteries while hot

spots 820a-820d correspond to the virtual patient's lungs. The sounds produced vary according to the state of the virtual patient. That is, the cardiac rhythms presented by the simulator can depend on the blood pressure, pulse, and age of the virtual patient.

Instead of "listening" to the virtual patient using the virtual stethoscope 810g, a user may choose to virtually percuss (i.e., simulate a gentle thumping of a patient's body) the virtual patient. Medical professionals often use percussion to identify abnormal masses. Selecting the percussion tool 810h may display a different collection of hot spots than those shown in FIG. 8. Selecting one of the hot spots causes the simulator to produce sounds corresponding to percussion of the corresponding area. Again, the sounds produced depend on the state of the virtual patient. For example, percussion of a healthy virtual patient's abdomen often produces a resonant echo while percussion of the abdomen of a virtual patient having a swollen liver may sound flatter.

As shown in FIG. 9, the interface palette also includes a fundascope/odoscope tool 810b for viewing images of the virtual patients eyes 902, retina 904, ears 906a-906b, nose (not shown), and throat 908. Descriptions 910 may accompany these images. Again, the virtual patient images 902-908 are not the same for each virtual patient, but are instead based on the virtual patient's current health. For example, a virtual patient suffering from retinopathy may cause display of a retinal image 904 depicting strands of white fibrous tissue? Similarly, a virtual patient suffering from an inner ear infection may cause display of an ear image 906a-906b depicting a reddened tympanic membrane.

As shown in FIG. 10, a user can view graphs 1000 of a virtual patient's vital statistics and lab tests over time.

The graphs 1000 enable the user to quickly grasp the effects their treatment has had on the virtual patient and identify correlations between movements of different charted variables.

As shown in FIG. 11, the simulator provides each virtual patient with a patient history 1100. The history 1100 may include clues to potential ailments. The history 1100 also may chronicle changes in the virtual patient's health over time.

As shown in FIG. 12, the interface 1200 enables a user to ask the virtual patient a question, for example, by typing questions into a text field 1204. Just as in real-life, questioning a patient can quickly yield information needed to formulate and confirm a diagnosis. The simulator generates a virtual patient response 1206 for each question asked 1204. Additionally, the simulator can adjust the facial expression 1202 of the virtual patient to portray a patient's emotional response to a question 1204. Potentially, the virtual patient's expression 1202 may be as telling as their actual response. As shown, the questions 1204 and answers 1206 take the form of text, however, in other embodiments, speech recognition and speech synthesis handle question input and answer output. Again, the response of the virtual patient can depend on state data. For example, the patient's response to "How's your vision" can differ for a healthy patient and a patient suffering from diabetes.

As shown in FIG. 13, the interface 1300 enables a user to make different interventions. For example, the user can make a diagnosis 1302. The simulator may provide feedback when a user correctly diagnoses a virtual patient ailment. The user may also order different lab tests 1306. The simulator returns lab results based on state data of the virtual

patient. The lab results can provide clues to ailments afflicting the virtual patient.

A user may also prescribe an over-the-counter or prescription medication 1304 or order a lifestyle change such as a change in diet. Such interventions can alter the state of the virtual patient. Of course, as in real-life, such interventions can have beneficial and/or problematic results and can cause predictable and unpredictable side-effects.

As shown in FIG. 14, the interface permits users to control "virtual time". That is, a user can speed up the simulation to see a long-term illness unfold in a matter of minutes or slow down the simulation to see a quick-striking illness slowly unfold.

FIGs. 8-14 illustrate an interface that provides users with techniques for monitoring and treating a virtual patient. The screenshots shown, however, are merely an example of one possible interface. A wide variety of other user interface presentations can take advantage of the techniques described herein.

#### B. Multimedia Presentation of a Virtual Patient

FIGs. 15A-15B show data structures 1502-1538 that control multimedia presentations provided by the interface. For example, the "Patient Views" data structure defines different patent model images. The multimedia fields MMFile1, MMFile2, MMFile3, and MMFile4 of the "Patient Views" data structure specify the file locations of previously generated model images corresponding to clothed-front-side, clothed-back-side, unclothed-front-side, and unclothed-back-side model views, respectively. These images correspond to controls 808a-808d in FIG. 8. Changing the files referred to can change the presentation of the virtual patient. For example, changing

the file associated with MMFile1 from aprime.jpg to slim\_prime.jpg can cause the user interface to present a slimmer clothed-front-side image of the virtual patient.

Similarly, the Chest "Front Auscultation" data structure 1510 defines multimedia presentations for use of the virtual stethoscope (see FIG. 8). Data structure elements Sound1, Sound2, Sound3, and Sound4 correspond to the sounds produced by selecting hot spots 816 in FIG. 8. Again, changing the sound files associated with Sound1, Sound2, Sound3, or Sound4 change the sound file played.

As shown, the data structures specify multimedia presentations for front auscultation 1506 and percussion 1508 of the virtual patient's abdomen, front 1510 and back 1512 auscultation of the virtual patient's chest, and percussion of the virtual patient's back 1514. The data structures also specify multimedia presentations for a virtual patient EKG 1516, examination of extremities 1518-1522, examination of head, eyes, ears, nose, and throat 1524-1530, neurological examination such as speech 1532 and gait 1534, and radiology images of the virtual patient's chest 1536 and skull 1538.

Though the data structures listed in FIGs. 15A and 15B include the term "objects", the data structures need not be implemented using an object-oriented methodology.

### C. Generating a Virtual Patient

In some embodiments, the simulator operates by interpreting data structures encoding different virtual patient characteristics. For example, a developer can code data structures that define the progression of different medical conditions. Procedures interpret the data structures to present a virtual patient. Interpretation of the data structures can include accessing and modifying virtual patient



state data. Such state data can include vital statistic variables (e.g., pulse rate, age, gender, weight, systolic blood pressure, diastolic blood pressure, respiration, and temperature), laboratory test variables, the state of  
5 different conditions, and other developer defined variables.

With some exceptions, described below, different virtual patient characteristics can proceed independently of one another. For example, different medical conditions may read and update the state data of the virtual patient and act  
10 without knowledge of other conditions. Thus, a virtual patient can suffer from a strange mix of symptoms caused by different conditions. To successfully treat the patient, the user learns to untangle the different symptoms into individual, treatable illnesses.

FIG. 16 shows a process 1600 for providing a virtual patient. Initially, a developer defines 1602 a domain of potential virtual patient characteristics. For example, the developer can define data structures for different ailments that evolve over time ("evolutions") that are computed by  
15 mathematical modeling of other variables ("computations"), for random fluctuations in variables ("migrations"), and for different treatments (e.g., drugs) a user can prescribe.

A patient generation process generates 1604 a virtual patient by selecting virtual patient characteristics from the domain of potential virtual domain characteristics. For  
25 example, the patient generator process 1604 may generate one patient that has the potential for diabetes and may later generate another patient that has the potential for osteoporosis. Patients are not limited to a single  
30 characteristic but can instead have different mixes of potential ailments.

Patient generation 1604 may feature interpretation of a patient generation script that specifies different probabilities of virtual patient characteristics occurring in the virtual patient. Patient generation 1604 may also  
5 initialize variables to specific values. The script may identify a particular medical area. For example, a script that generates a virtual patient having osteoporosis and cataracts may belong to a "geriatric" medical area. Additionally, the script may differ for men and women.

10 The script used to generate a patient may be randomly selected from all scripts. Alternatively, a user may specify a medical area and restrict patient generation to those scripts belonging to that area. Patient generation 1604, however, can occur without a script. For example, a patient  
15 generator can produce a virtual patient having a random selection of medical characteristics.

After generating 1604 the virtual patient, a simulator presents 1606 the virtual patient to the user based on virtual patient state data (e.g., vital statistic variables and lab  
20 test variables) and updates 1608 the state data by interpreting the data structures defining virtual patient characteristics.

FIG. 17 shows a diagram of a simulator 1700. In addition to the user interface 1722 (see FIGs. 8-14), the simulator  
25 1700 includes procedures 1701 that interpret data structures defining the virtual patient. The procedures 1701 handle questioning 1702 of the virtual patient (e.g., review of systems (ROS) questions), responding to lab test 1704 requests from the user, responding to physical examinations 1706  
30 requested by the user, and responding to orders 1708 issued by the user.

The procedures 1701 also modify state data of the virtual patient by interpreting evolution 1712 and migration 1714 data structures, and computation 1715 data structures. Evolutions 1712 control variables to reflect some condition. For  
5 example, a diabetes evolution will cause the virtual patient to gain weight. By contrast, migrations 1714 randomly vary variables within specified bounds. The combination of migrations 1714 and evolutions 1712 force a user to distinguish between significant and insignificant variations  
10 in vital signs and lab tests results. Computations 1715 allow mathematical modeling of system variables based on value(s) of other system variable(s).

As shown, the simulator 1700 also provides multimedia files 1716 such as video (e.g., MPEG (Motion Pictures Experts Group)) files, sound files, and picture (e.g., JPEG (Joint Photographers Experts Group)) files. These files may include instructional references material and/or multimedia features used to present the virtual patient. The state data 1718 can include references or copies of the multimedia files that  
15 reflect the current state of the patient (see FIGs. 15a and 15b).

Trigger and morphing procedures 1710 can update the multimedia files associated with the virtual patient based on logic operating on variables. For example, a trigger 1710  
20 procedure may change an image of a patient's torso from "normal\_torso.jpeg" to "slim\_torso.jpeg" if the patient's weight falls under one-hundred fifty pounds.

FIG. 18 depicts evolution 1800 of a medical condition. Evolutions alter variable values and/or lab test results to  
30 reflect some condition. The evolution 1800 has several modes: an initial mode 1802 before any events (e.g., treatments by the user and/or other state data) initiate the evolution, an

interim mode 1808 when the evolution temporarily lies dormant, and an active mode 1814 when the evolution changes 1818 the virtual patient's state data.

In addition to programming different events or criteria that start an evolution 1804 and the state data affected by an active 1814 condition, a programmer can identify different inhibiting events or criteria 1806, 1810 that halt progression of the evolution. For example, an order for "diet and exercise" may halt many different illnesses. Thus, identification of disease warning signs and prompt treatment can prevent the onset of a disease.

A programmer can also control the dominance 1816 of an evolution over other evolutions. For example, a first evolution may elevate blood pressure while a second evolution decreases blood pressure. If so desired, a programmer can code the first evolution to cancel the second evolution.

FIG. 19 shows a listing of statements defining an evolution 1900 for "Diet and Exercise". Such an evolution can regulate vital signs in a manner beneficial to a virtual patient's health. The evolution 1900 begins when a user orders "Diet and Exercise". After an interim period of four days 1906, the evolution begins decreasing a virtual patient's weight 1910a and glucose levels 1910b-1910d. While nothing can inhibit 1908 the effects of exercise, the evolution 1900, as shown, does not directly cancel any other evolutions 1908. While recommending "Diet and Exercise" does not directly cancel other evolutions, the evolution's 1900 control over weight and glucose variable values may inhibit a disease triggered by a particular weight or glucose level.

FIG. 20 shows a listing of statements that define a migration. Like an evolution, a migration adjusts the values of variables and other state data. Unlike an evolution the

migration adjusts the state data in a more random manner to represent ordinary fluctuations in measurements. As shown, the "Activated clotting time" migration maintains the value 2002 of the corresponding "ACT" variable between "114" and "186" from a starting point of "143". The migration operates once a "virtual" minute 2004. The migration is not purely random, but instead implements a "random walk" having changes that amount to, in this case, a 4% increase or decrease 2006 at most.

FIG. 21 shows a process 2100 for producing a trigger. Triggers implement conditional statements. Triggers can not affect variables but rather affect multimedia or text objects based on the values of variables. Typically, a programmer can use triggers to substitute different multimedia presentations of a virtual patient based on satisfaction of some criteria. As shown, the process 2100 tests for satisfaction of some criteria 2102. For example, the trigger may test the variables and/or the issuance of some order by the user (e.g., prescribing a drug). If satisfied, a trigger makes a specified alteration 2104.

FIG. 22 shows data structure statements 2202 that define the sounds associated with virtual percussion of the back of the virtual patient. Each sound corresponds to one of the hot spots (see FIG. 7). A trigger 2204 can change the sound of one of the hot spots. As shown, the trigger substitutes references to Wav Files/Percussion/percussThin.wav for the previous value of Sound1 ("PercusResonant.wav") if the weight of the patient falls between "100" and "120" pounds.

Like triggers, morphing can alter the image used to portray the virtual patient. FIG. 23 shows a flowchart 2300 of a process for morphing an image of a patient. Based on a specified maximum and minimum variable value, the process 2300

uses a current value of a variable to determine an offset into a video file, such as a file slowly morphing the image of a virtual patient from a slim to heavysset appearance. The process presents the image corresponding to the offset. The video file provides a fine gradation of images thus can present smooth and gradual changes in the virtual patient.

*ms*  
~~FIG. 24 shows a listing of statements defining a morph.~~  
The "weight" morph is based on the weight variable and extends from a minimum weight of 164 to a maximum weight of 184. Thus, if a patient weights "174", the corresponding offset would index halfway through a video file and present the corresponding image.

FIG. 25 shows a flowchart of a process 2500 that provides virtual patient responses to different questions posed by the user (see FIG. 12). After receiving a question 2502, the process searches 2502 for a corresponding entry in a collection of questions. For example, the process may identify key words in the question by eliminating articles and prepositions and search for a stored question having the identified key words. The keywords may also include Boolean criteria (e.g., this is the response when the query includes word a AND word b but NOT word c. The process 2500 may alternatively use other natural language processing techniques to match a user's question with a particular response. After finding an entry, the process presents 2506 the response corresponding to the question.

FIG. 26 shows a listing of statements used to define a "Review of Systems" response. The statements include a question, answer, keywords identifying the entry, an entry for the log, and a facial expression.

FIGs. 27-29 show data structures for lab tests, user orders, and physical examinations, respectively. FIG. 30

shows a data structure for a computation, where the MethodName "ComputeBPDiastolic" refers to a compiled function in a DLL.

A computation enables the simulator to use complex mathematical models of a virtual patient. The computation enables a programmer to specify condition statements (e.g., if (temperature > 100)) and/or relationships between variables (e.g., systolic pressure = diastolic pressure).

All components of the virtual patient simulator may reside on the same computer. In other embodiments, the virtual patient simulator uses the architecture shown in FIGs. 1 to 7 to provide different Internet users with a simulation of a patient medical exam.

Briefly, the simulator procedures 120 operate on different state data and different data structures for different virtual patients. Each virtual patient has an entry in a patient table 3110. Each virtual patient can correspond to a single application instance, though many different clients can simultaneously engage the same patient for group treatment, discussion, and education. A patient identifier may key each patient table entry 3110.

As shown the database 106 includes tables that track the current loads 3112 of different LAN virtual patient simulators, vital statistic variables 3120, lab variables 3126, pending physician orders 3116 and questions 3118 and their corresponding responses 3122, 3124. The database 106 can also store the data structures identifying the images and sounds 2904 that portray the virtual patient's current health.

### Embodiments

The techniques described here are not limited to any particular hardware or software configuration; they may find applicability in any computing or processing environment. The

techniques may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs executing on programmable computers that each include a processor, a storage medium  
5 readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code is applied to data entered using the input device to perform the functions described and to generate output information. The output  
10 information is applied to one or more output devices.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired.  
15 In any case, the language may be a compiled or interpreted language.

Each such computer program is preferable stored on a storage medium or device (e.g., CD-ROM, hard disk or magnetic diskette) that is readable by a general or special purpose  
20 programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer  
25 program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other embodiments are within the scope of the following claims.